Attention Networks and Diffusion Models
Introduction

Alexander Schönhuth

UNIVERSITÄT BIELEFELD
Faculty of Technology

Bielefeld University
April 11, 2023

Who are we?

- Research group "Genome Data Science"
  `https://gds.techfak.uni-bielefeld.de`
- Coordinates:
  Prof. Dr. Alexander Schönhuth
  email: aschoen@cebitec.uni-bielefeld.de
  office: UHG U10-128

Organization

Modules

- Lecture part of modules
  - *39-M-Inf-ABDA Advanced Big Data Analytics / Big Data Machine Learning* (graded, "benotete Prüfungsleistung")
    - See here `https://ekvv.uni-bielefeld.de/sinfo/publ/modul/308598306`
  - *24-M-P2 Profilierung 2* (ungraded, "Studienleistung")
    - See here `https://ekvv.uni-bielefeld.de/sinfo/publ/modul/27461022`

Presentation, Reports, Papers

- Presentations:
  - Individual presentations
  - To last for approx. 30 minutes, followed by discussion
  - Present contents of scientific paper
- Reports:
  - Reports summarize contents of paper
  - Reports 8-10 pages
- Papers:
  - Papers: some already available, list will be completed
  - Papers available via Wiki: `https://gds.techfak.uni-bielefeld.de/teaching/2023summer/attention`

Schedule

- Organization and introduction: *today*
- How to present (brief): *Apr 18* (online)
- How to write (brief): *Apr 25* (hybrid)

Schedule II

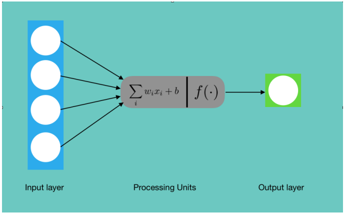
- **Presentations:** *from May 16* (earlier possible if desired)
    - Up to two presentations per week, if that suits everyone's schedules
    - If desired/necessary, block seminar day possible as well
- **Technical Report:** *after presentation:*
    - Optimally, report profits from feedback provided after presentation
    - Drafts can be submitted for discussion
    - Improving drafts based on feedback
    - *Submission deadline: July 31*

Attention Networks: Tutorial

Neural Networks

Neurons

Linear + Activation Function



$$\text{output} = a(w^T \cdot x + b)$$

Note: replace f in Figure by a!

Neuron: linear function followed by activation function

Examples

- Linear regression:

$$a = \text{Id}$$

a is identity function

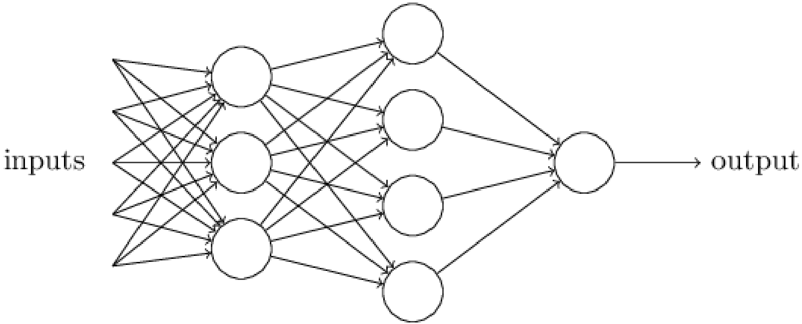
- Perceptron:

$$a(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

a is step function

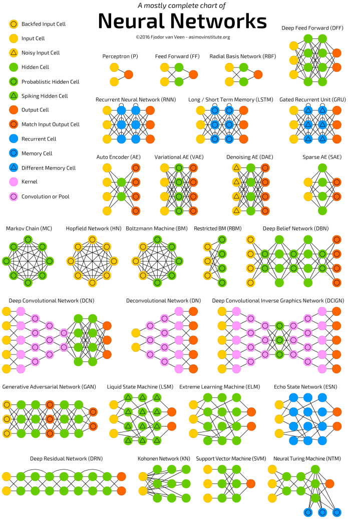
Neural Networks

Concatenating Neurons

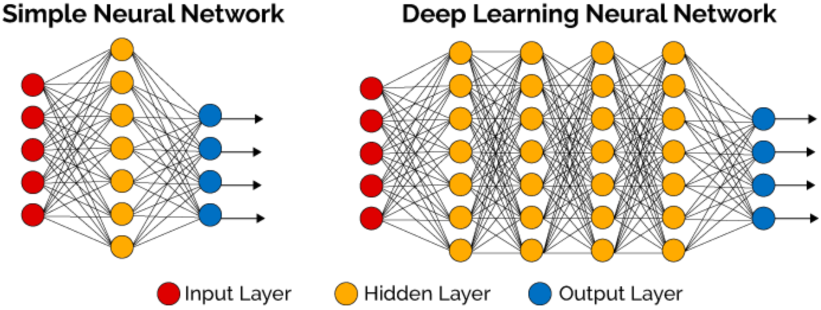


Neural Networks

Architectures



FEEDFORWARD NEURAL NETWORKS



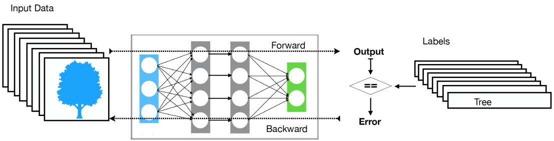
Width = Number of nodes in a hidden layer
Depth = Number of hidden layers
Deep = depth ≥ 8 (for historical reasons)

Feedforward Neural Networks

Formal Definition

- Let $\mathbf{x}^l \in \mathbb{R}^{d(l)}$ be all outputs from neurons in layer l, where $d(l)$ is the *width* of layer l.
- Let $y \in V$ be the output.
- Let $\mathbf{x} =: \mathbf{x}^0$ be the input.
- Then
$$\mathbf{x}^l = \mathbf{a}^l(\mathbf{W}^{(l)}\mathbf{x}^{l-1} + \mathbf{b}^l)$$
where $\mathbf{a}^l(.) = (a_1^l(.), ..., a_{d(l)}^l(.))$, $\mathbf{W}^{(l)} \in \mathbb{R}^{d(l) \times d(l-1)}$, $\mathbf{b}^l \in \mathbb{R}^{d(l)}$
- The function f representing a neural network with L layers (with depth L) can be written
$$y = f(\mathbf{x}^0) = f^{(L)}(f^{(L-1)}(...(f^{(1)}(\mathbf{x}^{(0)}))...))$$
where $\mathbf{x}^l = f^{(l)}(\mathbf{x}^{l-1}) = \mathbf{a}^{\mathbf{l}}(\mathbf{W}^{(\mathbf{l})}\mathbf{x}^{l-1} + \mathbf{b}^{\mathbf{l}})$

TRAINING: BACKPROPAGATION



- *E.g.* let X be a set of images, labels 1 and 0: tree or not
- Let
$$f_{(\mathbf{w},\mathbf{b})} : X \to \{0,1\} \quad \text{and} \quad \hat{f} : X \to \{0,1\}$$
network function ($f_{\mathbf{w},\mathbf{b}}$) and true function ($\hat{f}$)
- $L(f_{(\mathbf{w},\mathbf{b})}, \hat{f})$ loss function, differentiable in network parameters $\mathbf{w}$, $\mathbf{b}$
- *Back Propagation*: Minimize $L(f, \hat{f})$ through gradient descent
  - ☞ Heavily parallelizable!
- Decisive: Ratio number of parameters and training data

Why Neural Networks?

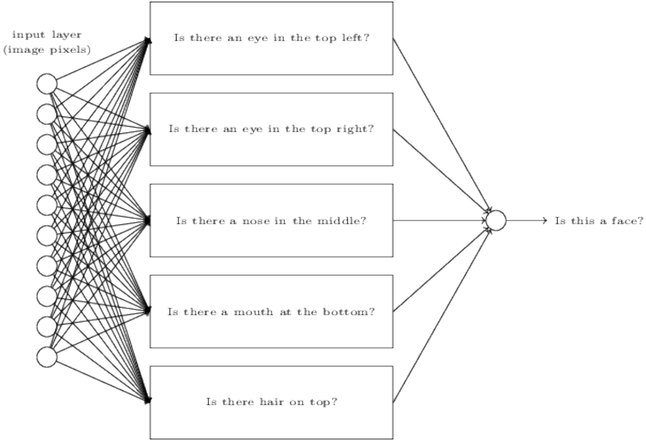
Why Neural Networks?

Given an (unknown) functional relationship $f : \mathbb{R}^d \to V$, why should we learn f by approximating it with a neural network?

Practical, Intuitive Consideration

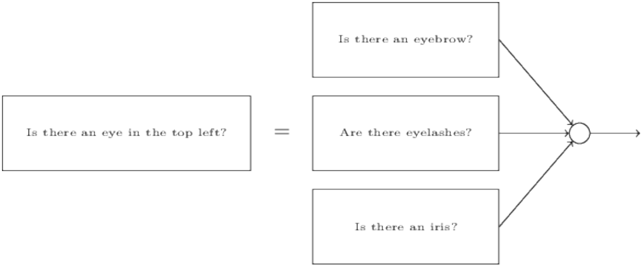
Deep Learning

Intuitive Explanation



- *Face recognition*: decompose classification task into subtasks

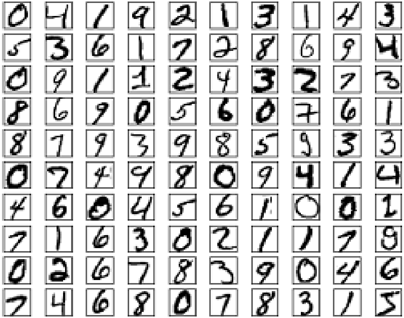
Deep Learning is Intuitive



- *Face recognition*: decompose subtask (eye recognition) into sub-subtasks
- Subtasks are composed into overall task "layer by layer"

Running Example: MNIST Classification

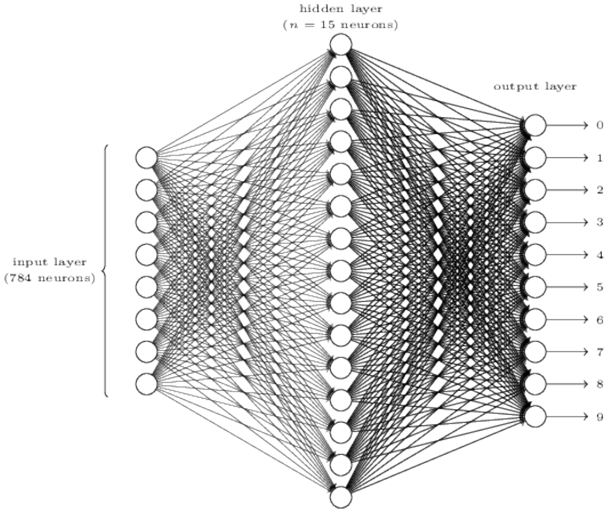
Data, Function



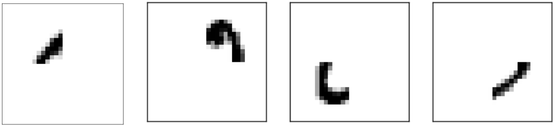
$$f : \mathbb{R}^{28 \times 28 = 784} \longrightarrow \{0, 1, ..., 9\} \tag{1}$$

Running Example

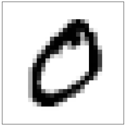
Model Class: NN with 1 hidden layer



Running Example



together makes



Neurons of hidden layer recognize characterizing parts of digit

Theoretical Consideration

The Universal Approximation Theorem

Theorem

A feedforward network with a single hidden layer containing a finite number of neurons can approximate any nonconstant, bounded and continuous function with arbitrary closeness, as long as there are enough hidden nodes.

Step function with n steps as neural network

- requires n hidden nodes
- hence $O(n)$ training data

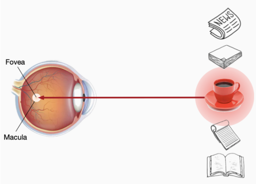
Attention

Biological Motivation

Attention: Motivation I

- Optic nerve receives 10^8 bits per second
- *Challenge:* Distinguish between important and irrelevant information
- *Solution: Attention*
  - Brain focuses on only a fraction of information
  - Smart usage of resources
  - Brain needs to know where to direct attention
- *Idea:* William James, “father of American psychology”, 1890’s
- Distinguish between *non-volitional* and *volitional cues*
  - They trigger subconscious and conscious actions

Attention: Nonvolitional Cues

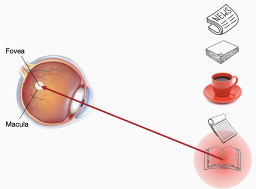


Nonvolitional cue: eye directs attention *non-voluntarily* to red coffee cup

From `https://d2l.ai`

- Nonvolitional cues based on saliency / conspicuity of objects
- *Example:*
  - Papers on desk black and white
  - Coffee cup red
  - *Consequence:* Eye "sees" coffee cup first
    ☞ Person grabs and drinks coffee

Attention: Volitional Cues



Deliberately searching for entertainment, eye *voluntarily* directs attention to book

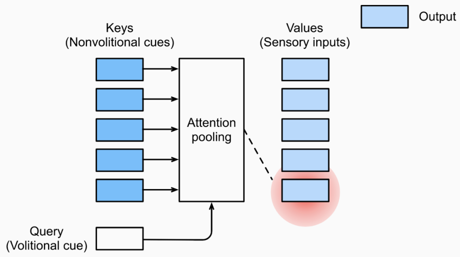
From `https://d2l.ai`

- Done with coffee, brain wants entertainment
- *Consequence:* Eye "sees" book in a deliberate attempt
- *Task-oriented search:*
  - Brain pre-trained to recognize objects that promise entertainment
  - Selection of book under full cognitive and volitional control

Queries, Keys and Values

Attention: Queries, Keys and Values I

Motivation

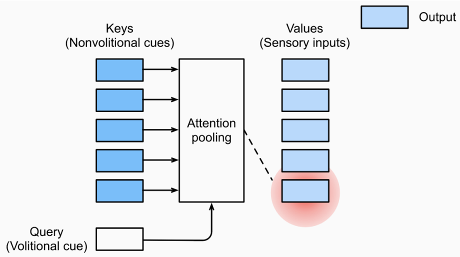


Attention pooling: integrating queries with keys (input) and values (output)

- There are no queries in feed forward neural networks
- Feedforward neural networks reflect non-volitional attention
- *Goal:* Model volitional attention cues and integrate them appropriately

Attention: Queries, Keys and Values II

Solution

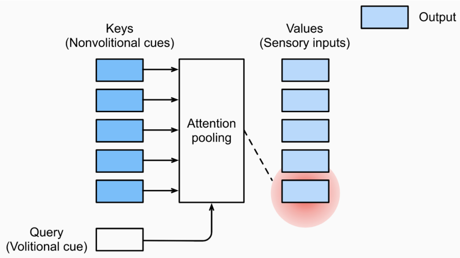


Attention pooling: integrating queries with keys (input) and values (output)

- Input / output ordinary neurons: *keys* and *values*
  - Keys and values come in pairs
- Volitional cues = *queries*
- Model patterned after database searches

ATTENTION: QUERIES, KEYS AND VALUES III

ATTENTION POOLING

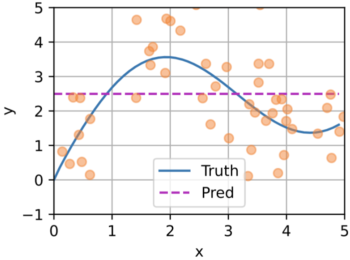


Attention pooling: integrating queries with keys (input) and values (output)

- *Attention weights* for keys reflect compatibility with query
- *Attention pooling:* Compute "attention weighted" sum of values
- *Output* dominated by values whose keys match query well

Attention Pooling

Attention Average Pooling



From `https://d2l.ai`

- *Truth:* $y = f(x) := 2\sin(x) + x^{0.8}$ (blue)
- *Data points* (x_i, y_i) sampled from $y_i = f(x_i) + \epsilon$ where ϵ follows normal distribution with $\mu = 0, \sigma = 0.5$ (orange dots)
- *Prediction:* $\hat{f}(x) := \sum_{i=1}^{n} y_i$ where $n = \#$ training data (dashed pink)
  - Reflects unweighted average pooling
- *Conclusion:* Unweighted average pooling not necessarily good idea

Nadaraya-Watson Kernel Regression I

- Let $K(.)$ be a *kernel*
- *Kernel properties:*
  - $K(x) \to 0$ for $|x| \to \infty$
  - $K(0)$ is maximum
- *Example: Gaussian kernel*

$$K(u) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{u^2}{2}) \tag{2}$$

- *Nadaraya-Watson kernel regression:* For unseen x, determine

$$\hat{f}(x) = \sum_{i=1}^{n} \frac{K(x - x_i)}{\sum_{j=1}^{n} K(x - x_j)} y_i \tag{3}$$

where $(x_i, y_i), i = 1, ..., n$ are the training data points

UNIVERSITÄT
BIELEFELD

Nadaraya-Watson Kernel Regression II

- *Nadaraya-Watson kernel regression:* For unseen x, determine

$$\hat{f}(x) = \sum_{i=1}^{n} \frac{K(x - x_i)}{\sum_{j=1}^{n} K(x - x_j)} y_i \tag{4}$$

  where $(x_i, y_i), i = 1, ..., n$ are the training data points

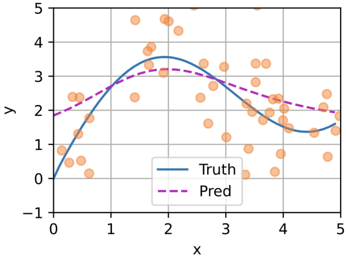
- This agrees with general concept of attention pooling

$$\hat{f}(x) = \sum_{i=1}^{n} \alpha(x, x_i) y_i \tag{5}$$

  where x is query, and (x_i, y_i) are key-value pairs

- Value y_i receives more weight the closer its key x_i to x

Nadaraya-Watson Kernel Regression III

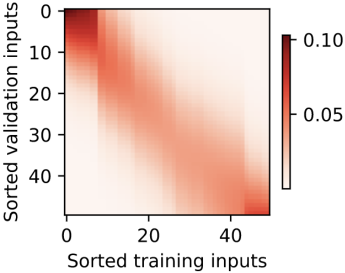


From `https://d2l.ai`

- Plugging the Gaussian kernel (2) into (4),(5) yields (dashed pink curve)

$$
\begin{aligned}
\hat{f}(x) &= \sum_{i=1}^{n} \alpha(x, x_i) y_i = \sum_{i=1}^{n} \frac{\exp(-\frac{1}{2}(x - x_i)^2)}{\sum_{j=1}^{n} \exp(-\frac{1}{2}(x - x_j)^2)} y_i \\
&= \sum_{i=1}^{n} \operatorname{softmax}(-\frac{1}{2}(x - x_i)^2) y_i
\end{aligned} \tag{6}
$$

Nadaraya-Watson Kernel Regression IV



From https://d2l.ai

- 50 training data points (x_i, y_i)
- 50 validation data points x
- Sort training and validation data by x_i and x resp.
- Plot $\alpha(x, x_i) = \sum_{i=1}^{n} \text{softmax}(-\frac{1}{2}(x - x_i)^2)$ for each pair (x_i, x)

Nadaraya-Watson Kernel Regression V

- Nadaraya-Watson kernel regression

$$\hat{f}(x) = \sum_{i=1}^{n} \alpha(x, x_i) y_i = \sum_{i=1}^{n} \frac{K(x - x_i)}{\sum_{j=1}^{n} K(x - x_j)} y_i \tag{7}$$

  is an example of *nonparametric attention pooling*
- *Benefit:* Converges to true function on increasing training data
  - *Reminder:* Training data reflect key-value pairs
- *Disadvantage:* There are no learnable parameters

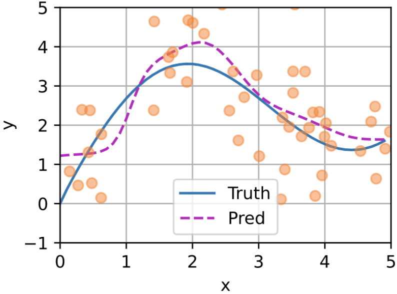
Parametric Attention Pooling I

- Integration of a learnable parameter w into (6) yields

$$\begin{aligned}\hat{f}(x) &= \sum_{i=1}^{n} \alpha(x, x_i) y_i = \sum_{i=1}^{n} \frac{\exp(-\frac{1}{2}((x - x_i)w)^2)}{\sum_{j=1}^{n} \exp(-\frac{1}{2}((x - x_j)w)^2)} y_i \\ &= \sum_{i=1}^{n} \text{softmax}(-\frac{1}{2}((x - x_i)w)^2) y_i\end{aligned} \quad (8)$$

- The parameter w can be learnt via (stochastic) gradient descent
- w reflects influence span of keys on queries
  - Number of influential keys decreases on increasing w

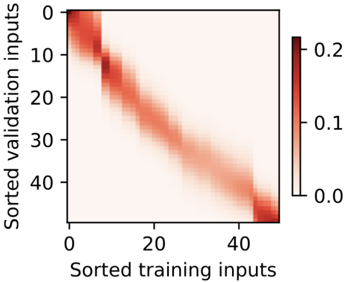
Parametric Attention Pooling II



From `https://d2l.ai`

- Predicted curve is less smooth than nonparametric counterpart

Parametric Attention Pooling III



From `https://d2l.ai`

- Training / validation procedure analogous to nonparametric setting
- However, training includes learning parameter w
- Region with larger attention weights sharper in parametric setting

Attention Scoring Functions

Attention Pooling: Digest I

- Re-consider (6):

$$\hat{f}(x) = \sum_{i=1}^{n} \alpha(x, x_i) y_i = \sum_{i=1}^{n} \text{softmax}(-\frac{1}{2}(x - x_i)^2) y_i$$

- One can view $\alpha(x, x_i)$ as
  - an attention scoring function

$$a(x, x_i) := -\frac{1}{2}(x - x_i)^2 \tag{9}$$

  - that is further fed into a softmax operation, yielding

$$\alpha(x, x_i) = \text{softmax}(a(x, x_i)) \tag{10}$$

Attention Pooling: Digest II

- One can view $\alpha(x, x_i)$ as
  - an attention scoring function

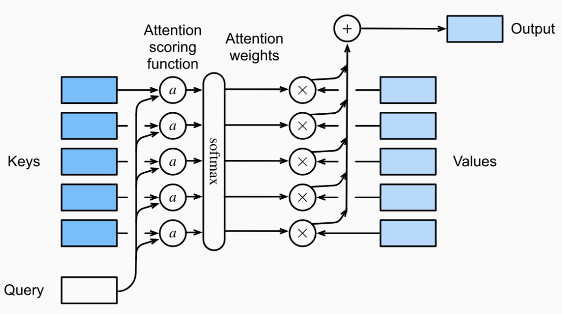
$$a(x, x_i) := -\frac{1}{2}(x - x_i)^2 \tag{11}$$

  - that is further fed into a softmax operation, yielding

$$\alpha(x, x_i) = \text{softmax}(a(x, x_i)) \tag{12}$$

- *Result:* Probability distribution
  - over values y_i paired with keys x_i where
  - probabilities are attention weights $\alpha(x, x_i)$

ATTENTION SCORING FUNCTIONS: MOTIVATION



Output of attention pooling is weighted average of values

- Let x be query, and x_i keys. Attention weights generally compute as

$$\alpha(x, x_i) = \text{softmax}(a(x, x_i)) \tag{13}$$

- *Advantage:* Freedom in choosing attention scoring functions $a(x, x_i)$

Attention Pooling: Formal Summary

- Let $\mathbf{q} \in \mathbb{R}^q$ be a query and $(\mathbf{k}_1, \mathbf{v}_1), ..., (\mathbf{k}_m, \mathbf{v}_m), \mathbf{k}_i \in \mathbb{R}^k, \mathbf{v}_i \in \mathbb{R}^v$ be m key-value pairs
- The *attention pooling* f computes as

$$f(\mathbf{q}, (\mathbf{k}_1, \mathbf{v}_1), ..., (\mathbf{k}_m, \mathbf{v}_m)) = \sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i \in \mathbb{R}^v \tag{14}$$

- The *attention weight* $\alpha(\mathbf{q}, \mathbf{k}_i) \in \mathbb{R}$ computes as

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \text{softmax}(a(\mathbf{q}, \mathbf{k}_i)) = \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^{m} \exp(a(\mathbf{q}, \mathbf{k}_j))} \tag{15}$$

- The *attention scoring function* $a(\mathbf{q}, \mathbf{k})$ maps two vectors to a scalar

$$a : \mathbb{R}^q \times \mathbb{R}^k \longrightarrow \mathbb{R} \tag{16}$$

Additive Attention Scoring

- Let $\mathbf{q} \in \mathbb{R}^q$ be a query and $\mathbf{k} \in \mathbb{R}^k$ be a key
- Let $\mathbf{W}_q \in \mathbb{R}^{h \times q}, \mathbf{W}_k \in \mathbb{R}^{h \times k}, \mathbf{w}_v \in \mathbb{R}^h$ collect learnable parameters
- The *additive attention scoring function* computes as

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{w}_v^T \tanh(\mathbf{W}_q \mathbf{q} + \mathbf{W}_k \mathbf{k}) \in \mathbb{R} \tag{17}$$

- *Interpretation:* (17) reflects running $\mathbf{q}, \mathbf{k}$ through MLP
  - *Input:* Concatenation of $\mathbf{q}$ and $\mathbf{k}$
  - One *hidden layer* of width h
  - Parameters from input to hidden layer are $\mathbf{W}_q, \mathbf{W}_k$
  - The activation function is tanh
  - Parameters from hidden to output layer captured by $\mathbf{w}_v$

Scaled Dot-Product Attention Scoring

- Let $\mathbf{q}, \mathbf{k} \in \mathbb{R}^d$ be *equal-sized* query and key
- The *scaled dot-product attention scoring function* computes as

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^T \mathbf{k} / \sqrt{d} \tag{18}$$

- *Note:* Dot product $\mathbf{q}^T \mathbf{k}$ has mean 0 and variance d
  ☞ Dividing by $\sqrt{d}$ implies standard deviation of 1

Minibatches:

- Computing attention for n queries and m keys at once
- For queries $\mathbf{Q} \in \mathbb{R}^{n \times d}$, keys $\mathbf{K} \in \mathbb{R}^{m \times d}$, values $\mathbf{V} \in \mathbb{R}^{m \times v}$ compute

$$\text{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}})\mathbf{V} \in \mathbb{R}^{n \times v} \tag{19}$$

Thanks for your attention!